ECLiPSe by Example

www.eclipse-clp.org

Tutorial CP'07

Joachim Schimpf and Kish Shen

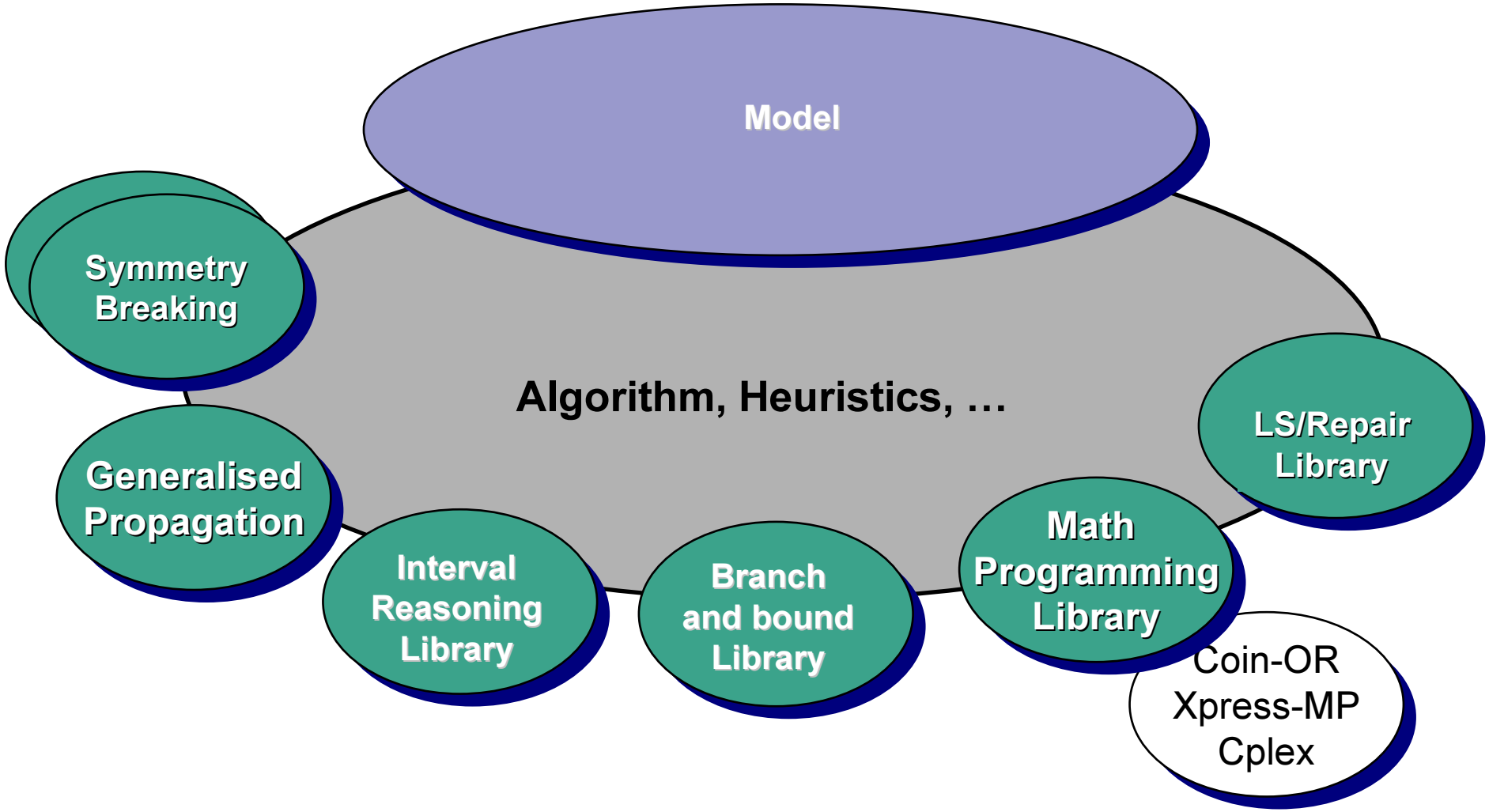
Motivation

ECLiPSe attempts to support - in some form or other - the most common techniques used in solving Constraint (Optimization) Problems:

- CP – Constraint Programming
- MP – Mathematical Programming
- LS – Local Search
- and combinations of those

ECLiPSe is built around the CLP (Constraint Logic Programming) paradigm

ECLiPSe for Modelling and Solving



ECLiPSe Usage

Applications

- Developing problem solvers
- Embedding and delivery

Research

- Teaching
- Prototyping solution techniques

ECLiPSe is open source (MPL)

- can be freely used for any purpose

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do Local Search
- How to use LP/MIP
- How to do hybrids
- How to visualise

ECLiPSe Programming Language (I)

- Logic Programming based

Predicates over Logical Variables	X #> Y, integers([X,Y])
Disjunction via backtracking	X=1 ; X=2
Metaprogramming (e.g. constraints as data)	Constraint = (X+Y)

- Modelling extensions

Arrays	M[I,J]
Structures	task{start:S}
Iteration/Quantification	(foreach(X,Xs) do …)

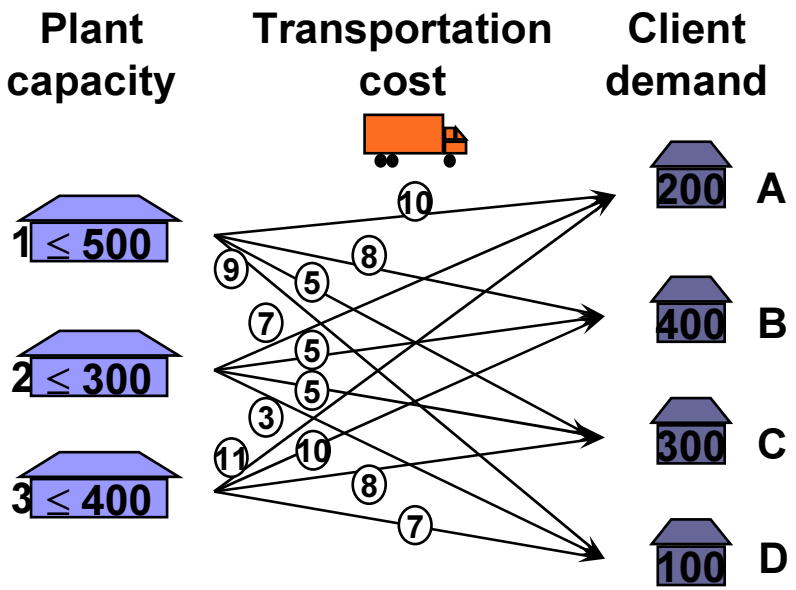
- Solver annotations

Solver libraries	:- lib(ic).
Solver qualification	[Solvers] : Constraint

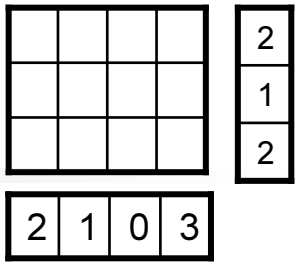
One language for modelling, search, and solver implementation!

Modelling
Solver independent model

```
model(Vars, Obj) :-

    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,

    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,

    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,

    Obj =
        10*A1 + 7*A2 + 11*A3 +
         8*B1 + 5*B2 + 10*B3 +
         5*C1 + 5*C2 +  8*C3 +
         9*D1 + 3*D2 +  7*D3.
```



Modelling
With Iterators and Arrays




```
model(RowSums, ColSums, Board) :-

   dim(RowSums, [M]),                          % get dimensions
   dim(ColSums, [N]),

   dim(Board, [M,N]),                          % make variables
   Board[1..M,1..N] :: 0..1,                   % domains

   ( for(I,1,M), param(Board,RowSums,N) do      % row cstr
       sum(Board[I,1..N]) #= RowSums[I]
   ),
   ( for(J,1,N), param(Board,ColSums,M) do      % col cstr
       sum(Board[1..M,J]) #= ColSums[J]
   ).
```

Modelling
With Structures and Predicates

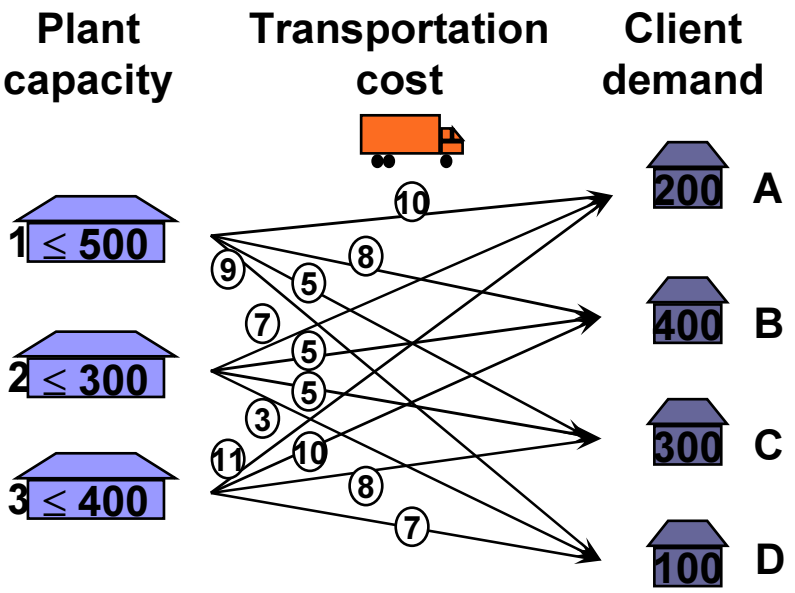
```
:- local struct(task(start,dur,resource,name)).

model(Tasks) :-
        …
        Task7 = task{start:S7,dur:7,name:"roof",res:R3},
        …
        precedes(Task7, Task3),
        …

precedes(task{start:S1,dur:D}, task{start:S2}) :-
        S2 #>= S1+D1.
```

Constraint Solver Libraries

Solver Lib	Var Domains	Constraints class	Behaviour
suspend	numeric	Arbitrary arithmetic in/dis/equalities	Passive test
fd	integer, symbol	Linear in/dis/equalities and some others	Domain propagation
ic 	real, integer	Arbitrary arithmetic in/dis/equalities	Bounds/domain propagation
ic_global	integer	N-ary constraints over lists of integers	Bounds/domain propagation
ic_sets	set of integer	Set operations (subset, cardinality, union, …)	Set-bounds propagation
ic_symbolic	ordered symbols	Dis/equality, ordering, element, …	Bounds/domain propagation
sd	unordered symbols	Dis/equality, alldifferent	Domain propagation
propia 	Inherited	any	various
eplex 	real, integer	Linear in/equalities	Global, optimising
tentative 	open	open	Violation monitoring

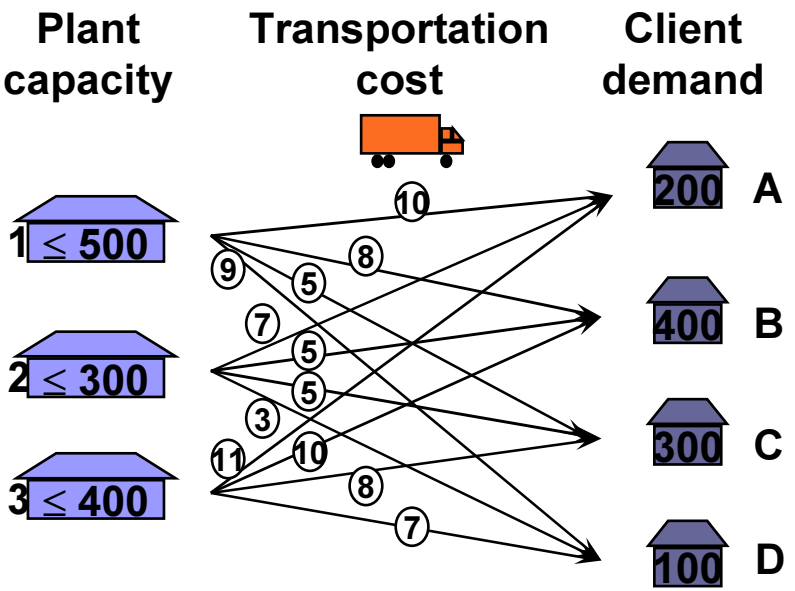
Solvers
Solving with Finite Domains

```
:- lib(ic).
:- lib(branch_and_bound).

solve(Vars, Cost) :-
    model(Vars, Obj),
    Cost #= eval(Obj),
    minimize(search(Vars, 0, first_fail, indomain_split, complete, []), Cost).

model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
         8*B1 + 5*B2 + 10*B3 +
         5*C1 + 5*C2 +  8*C3 +
         9*D1 + 3*D2 +  7*D3.
```



Solvers
Solving with Linear Programming

```
:- lib(eplex).

solve(Vars, Cost) :-
    model(Vars, Obj),
    eplex_solver_setup(min(Obj)),
    eplex_solve(Cost).


model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
         8*B1 + 5*B2 + 10*B3 +
         5*C1 + 5*C2 +  8*C3 +
         9*D1 + 3*D2 +  7*D3.
```



Common Arithmetic Solver Interface

Solver	$::/2 $=/2, =:=/2 $>=/2, >=/2 $=</2, =</2	$>/2, >/2 $</2, </2	$\=/2 =\=/2	::/2	#::/2 #= /2 #>=/2, #>/2 #=</2, #</2	#\=/2	integers/1	reals/1
suspend	✓	✓	✓	✓	✓	✓	✓	✓
ic	✓	✓	✓	✓	✓	✓	✓	✓
eplex	✓			✓			✓	✓
std arith	✓	✓	✓					

Solvers
The real/integer domain solver lib(ic)

Differences from a plain finite domain solver:

- Real-valued variables
- Integrality is a constraint
- Infinite domains supported
- Subsumes finite domain functionality

Solvers – interval solver lib(ic)
The basic set of constraints

Types

```
reals(Xs), integers(Ys)
```

Domains

```
X :: [1..5,8], Y :: -0.5..5.0, Z :: 0.0..inf
```

Non-strict inequalities

```
X #>= Y, Y #=< Z, X $>= Y, Y $=< Z
```

Strict inequalities

```
X #> Y, Y #< Z, X $> Y, Y $< Z
```

Equality and disequality

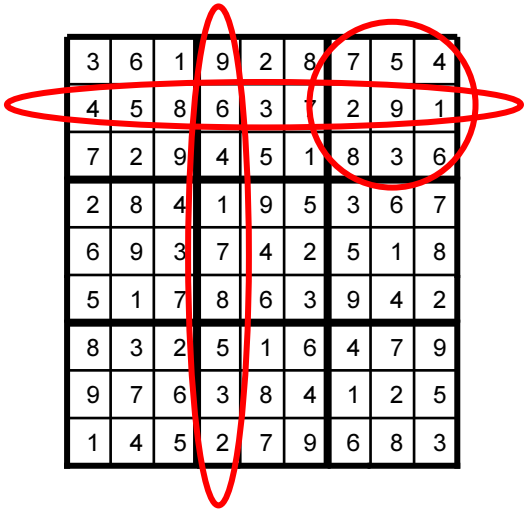
```
X #= Y, Y #\= Z, X $= Y, Y $\= Z
```

Expressions

```
+ - * / ^ abs sqr exp ln sin cos min max sum ...
```

"#" constraints impose integrality, "$" constraints do not

Solvers – interval solver lib(ic)
Pure finite domain problem




```
sudoku(Board) :-

    dim(Board, [9,9]),
    Board[1..9,1..9] :: 1..9,

    ( for(I,1,9), param(Board) do
        alldifferent(Board[I,1..9]),
        alldifferent(Board[1..9,I])
    ),
    ( multifor([I,J],1,9,3), param(Board) do
        ( multifor([K,L],0,2), param(Board,I,J), foreach(X,SubSquare) do
            X is Board[I+K,J+L]
        ),
        alldifferent(SubSquare)
    ),

    labeling(Board).
```

CP functionality – library(ic)
Mixing integer and continuous variables

From rectangular sheets that come in widths of 50, 100 or 200 cm, and lengths of 2,3,4 or 5 m, build the smallest cylinder with at least 2 m^3 volume [A&W]:

```
cylinder(W, L, V) :-
        W :: [50, 100, 200],           % width in cm
        L :: 2..5,                     % length in m
        V $>= 2.0,                     % min volume
        V $= (W/100)*(L^2/(4*pi)),
        minimize(labeling([W,L]), V).
```

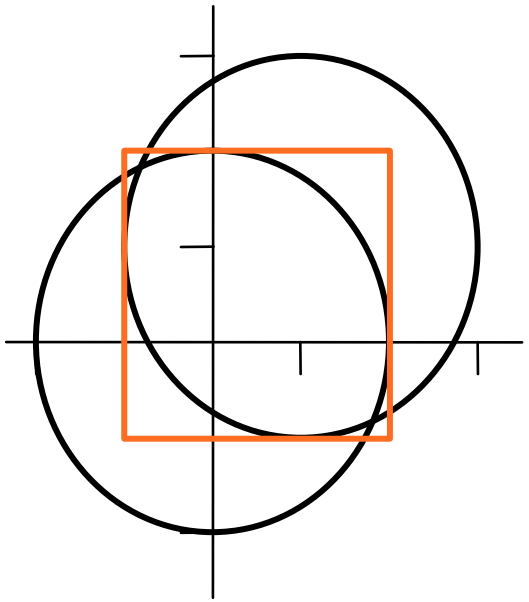
```
?- cylinder(W, L, V).
Found a solution with cost 2.546479089470325__2.546479089470326
Found no solution with cost 2.546479089470325 .. 1.546479089470326
W = 200
L = 4
V = 2.546479089470325__2.546479089470326
There are 6 delayed goals.
Yes (0.00s cpu)
```

"Bounded real" result

Conditional solution
Due to limited precision

CP functionality – library(ic) Continuous variables

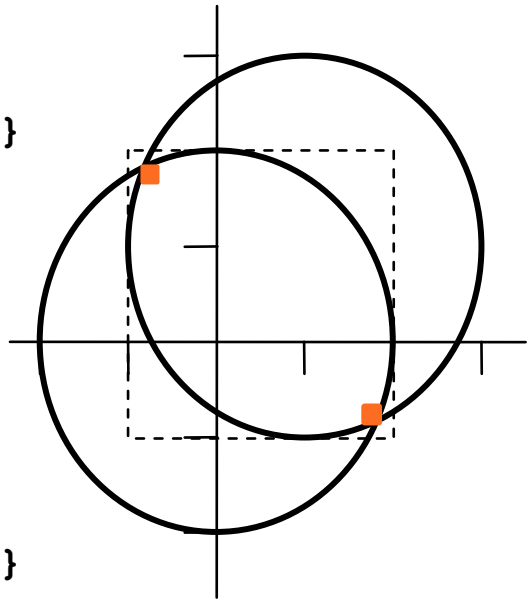
- Find the intersection of two circles



```
?- 4 $= X^2 + Y^2,
   4 $= (X - 1)^2 + (Y - 1)^2).

X = X{-1.000000000000002 .. 2.0000000000000013}
Y = Y{-1.000000000000002 .. 2.0000000000000013}

There are 12 delayed goals.
Yes
```

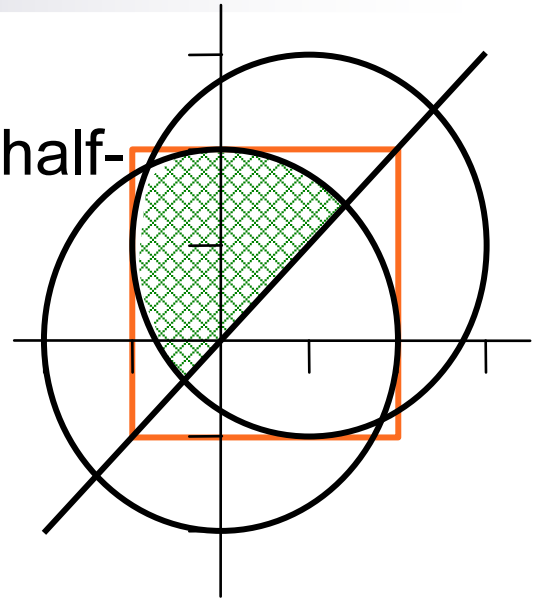
CP functionality – library(ic)
Isolating solutions via search

```
?- 4 $= X^2 + Y^2,
   4 $= (X - 1)^2 + (Y - 1)^2,
   locate([X, Y], 1e-5).

X = X{-0.8228756603527082 .. -0.82287564484819 9}
Y = Y{1.8228756448481993 .. 1.8228756603552705}

There are 12 delayed goals.
More ? ;

X = X{1.8228756448481993 .. 1.8228756603552705}
Y = Y{-0.8228756603527082 .. -0.82287564484819 9}

There are 12 delayed goals.
Yes
```



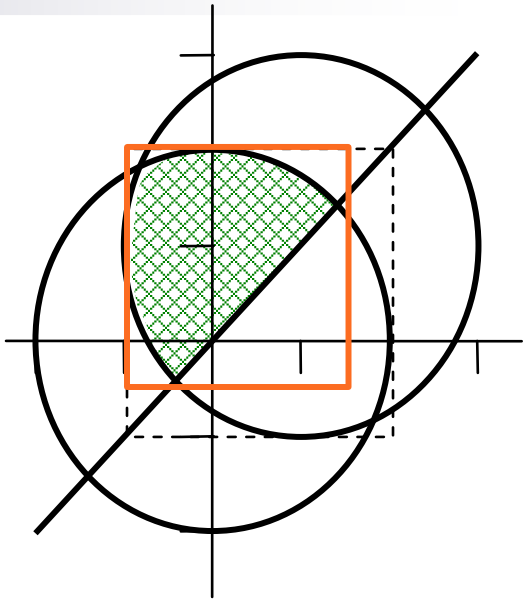
CP functionality – library(ic)
Continuous problems with feasible regions

- Find the intersection of two discs and a half-plane



```
?- 4 $>= X^2 + Y^2,
   4 $>= (X - 1)^2 + (Y - 1)^2,
   Y $>= X.

Y = Y{-1.000000000000002 .. 2.000000000000013}
X = X{-1.000000000000002 .. 2.000000000000013}

There are 13 delayed goals.
Yes
```

CP functionality – library(ic)
Tightening bounds by shaving



```
?- 4 $>= X^2 + Y^2,
   4 $>= (X - 1)^2 + (Y - 1)^2,
   Y $>= X,
   squash([X, Y], 1e-5, lin).

X = X{-1.000000000000002 .. 1.4142135999632603}
Y = Y{-0.4142135999632607 .. 2.000000000000013}

There are 13 delayed goals.
Yes
```

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do LS
- How to use LP/MIP
- How to do hybrids
- How to visualise

User-defined Constraints Using reification

Connecting primitives in reified form, combining booleans:

```
#=<(X+7, Y, B1), #=<(Y+7, X, B2), B1+B2 #>= 1.
```

The same with syntactic sugar:

```
X+7 #=< Y  or  Y+7 #=< X.
```

Clever example:

```
lex_le(Xs, Ys) :-
    ( foreach(X,Xs), foreach(Y,Ys), fromto(1,Bi,Bj,1) do
          Bi #= (X #< Y + Bj)
    ).
```

User-defined constraints
Generalised Propagation – lib(propia)

A generic algorithm to extract and propagate the MSG (most specific generalisation) from disjunctions [LeProvost&Wallace].

Syntax: NonDetGoal ***infers*** Spec

```
c(1,2).  c(1,3).  c(3,4).    % extensional constraint spec

?- c(X,Y) infers ic.
X = X{[1, 3]}
Y = Y{2 .. 4}
There is 1 delayed goal.

?- c(X,Y) infers ic, X = 3.
Y = 4
Yes.
```

User-defined constraints - Generalised Prop Constructive disjunction

```
?- [A, B] :: 1 .. 10, (A + 7 #=< B ; B + 7 #=< A) infers ic.
A = A{[1 .. 3, 8 .. 10]}
B = B{[1 .. 3, 8 .. 10]}
There is 1 delayed goal.
Yes (0.00s cpu)
```

Note the difference with reification:

```
?- [A, B] :: 1 .. 10,  A + 7 #=< B or B + 7 #=< A.
A = A{1 .. 10}
B = B{1 .. 10}
There are 3 delayed goals.
Yes (0.00s cpu)
```

User-defined constraints – Generalised Prop Prototyping AC and SAC constraints

Arc consistency from weaker consistency (test, forward checking)

```
ac_constr(Xs) :-
      (
               weak_constr(Xs),
               delete(X, Xs, Others),
               indomain(X),
               once labeling(Others)
      ) infers ic.
```

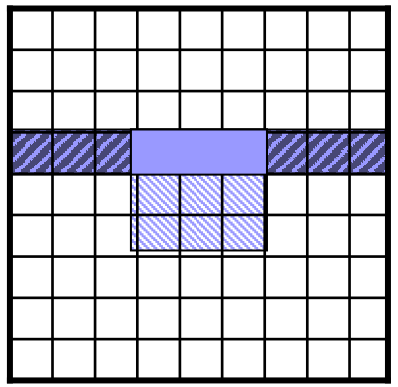
Singleton arc consistency from arc consistency:

```
sac_constr(Xs) :-
      (
               ac_constr(Xs),
               member(X, Xs),
               indomain(X)
      ) infers ic.
```

User-defined constraints – Generalised Prop Prototyping constraints

Or something weaker,

e.g. for combining constraints.

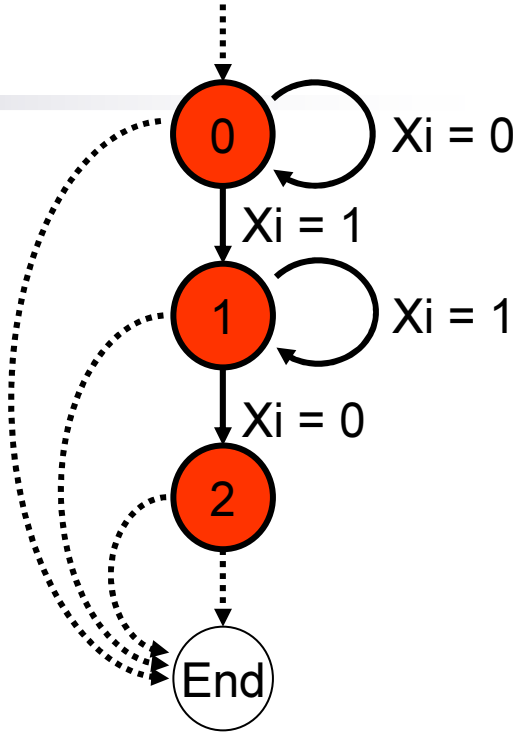


E.g. a constraint for sudoku:

```
overlapping_alldifferent(Xs, Ys, XYs) :-
    (
        alldifferent(Xs), alldifferent(Ys),
        labeling(XYs)
    ) infers ic.
```

User-defined constraints – Generalised Prop Graph/automaton method

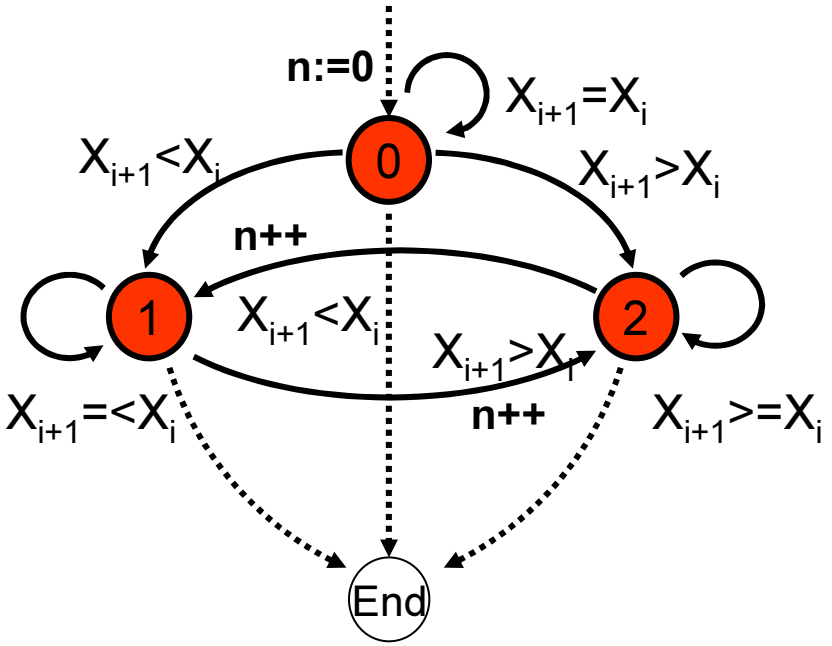
- Beldiceanu et al, 2004:
  Deriving Filtering Algorithms from Constraint Checkers




```
global_contiguity(Xs) :-

    StateEnd :: 0..2,
    (
        fromto(Xs, [X|Xs1], Xs1, []),
        fromto(0, StateIn, StateOut, StateEnd)
    do
       (
          StateIn = 0, (X = 0, StateOut = 0 ; X = 1, StateOut = 1 )
          ;
          StateIn = 1, (X = 0, StateOut = 2 ; X = 1, StateOut = 1 )
          ;
          StateIn = 2, X = 0, StateOut = 2

        ) infers ac
    ).
```

User-defined constraints – Generalised Prop Graph/automaton method (II)

```
inflexion(N, Xs) :-
    StateEnd :: 0..2,
    (
        fromto(Xs, [X1,X2|Xs1], [X2|Xs1], [_]),
        foreach(Ninc, Nincs),
        fromto(0, StateIn, StateOut, StateEnd)
    do
        (X1 #< X2) #= (Sig #= 1),
        (X1 #= X2) #= (Sig #= 2),
        (X1 #> X2) #= (Sig #= 3),
        ( StateIn = 0,
              ( Sig=1, Ninc=0, StateOut=1
              ; Sig=2, Ninc=0, StateOut=0
              ; Sig=3, Ninc=0, StateOut=2 )
        ; StateIn = 1,
              ( Sig=1, Ninc=0, StateOut=1
              ; Sig=2, Ninc=0, StateOut=1
              ; Sig=3, Ninc=1, StateOut=2 )
        ; StateIn = 2,
              ( Sig=1, Ninc=1, StateOut=1
              ; Sig=2, Ninc=0, StateOut=2
              ; Sig=3, Ninc=0, StateOut=2 )
        ) infers ac
    ),
    N #= sum(Nincs).
```



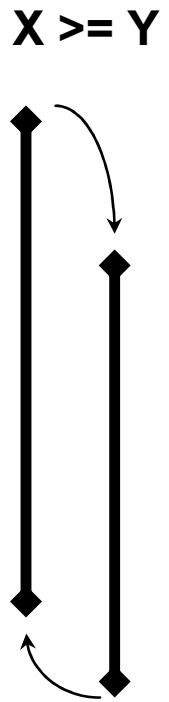
User-defined constraints
Using low-level primitives

Primitives for implementing propagators:

- Goal suspend/wake mechanism
- Variable-related triggers
- Execution priorities
- Solver's reflection primitives

E.g. bounds-consistent greater-equal:

```
ge(X, Y) :-
      ( var(X),var(Y) ->
          suspend(ge(X,Y), 3, [X->ic:max,Y->ic:min])
      ;
          true
      ),
      get_max(X, XH),
      get_min(Y, YL),
      impose_min(X, YL),
      impose_max(Y, XH).
```



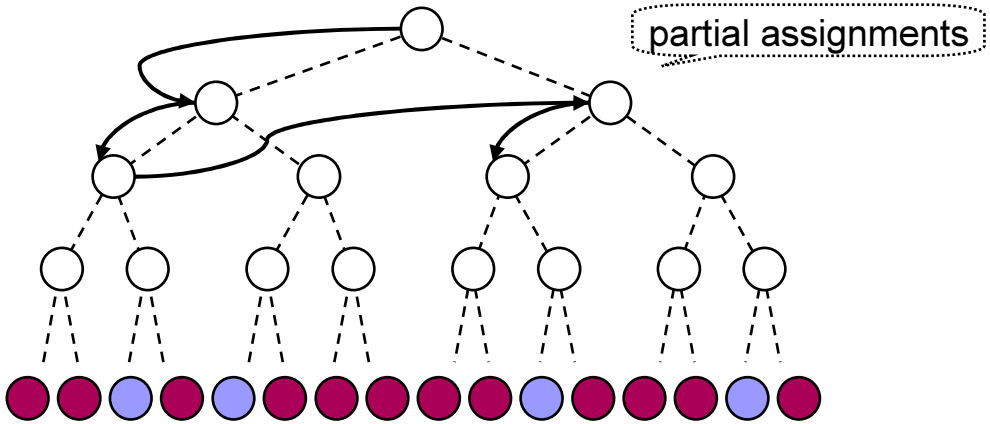
User-defined constraints
Single-propagator max-constraint

```
mymax(A, B, M):-
        get_bounds(A, MinA, MaxA),
        get_bounds(B, MinB, MaxB),
        get_bounds(M, MinM, MaxM),
        ( MinA >= MaxB ->
              A = M
        ; MinB >= MaxA ->
              B = M
          ; MinM > MaxB ->
              A = M
          ; MinM > MaxA ->
              B = M
          ;
              Max is max(MaxA, MaxB),
              Min is max(MinA, MinB),
              impose_bounds(M, Min, Max),
              impose_max(A, MaxM),
              impose_max(B, MaxM),

              Vars = [A,B,M],
              ( nonground(2, Vars, _) ->
                  suspend(mymax(A, B, M), 3, [Vars->ic:max,Vars->ic:min])
              ;
                  true
              )
          ).
```

Overview

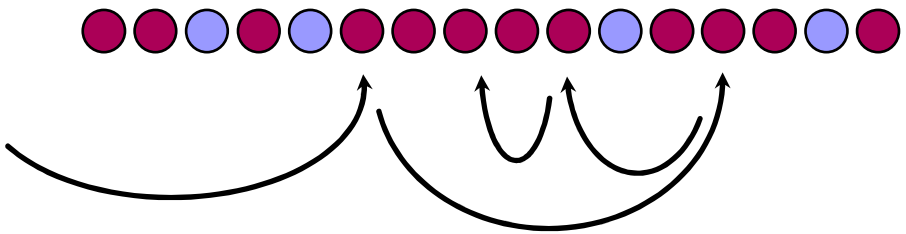
- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do LS
- How to use LP/MIP
- How to do hybrids
- How to visualise

Exploring search spaces



CLP Tree search:

- constructive
- partial/total assignments
- systematic
- complete or incomplete



“Local” search:

- move-based (trajectories)
- only total assignments
- usually random element
- incomplete

Tree Search
Labeling heuristics

For finite domains, common heuristics are provided by built-in:

search(List, VarIndex, VarSelect, ValSelect, SearchMethod, Options)

But often user-programmed, e.g. most basic:

```
labeling(AllVars) :-
      ( foreach(Var, AllVars) do
          indomain(Var)                    % choice here
      ).
```

Extends into schema for further heuristics:

```
labeling(AllVars) :-
      static_preorder(AllVars, OrderedVars),
      ( fromto(OrderedVars, Vars, RestVars, []) do
          select_variable(X, Vars, RestVars),
          select_value(X, Value),          % choice here
          X = Value
      ).
```

Tree Search
Symbolic manipulation for heuristics

- **Standard heuristics predefined (first-fail etc)**
- **Problem-specific heuristics programmable via reflection and symbolic manipulation features of the CLP language**

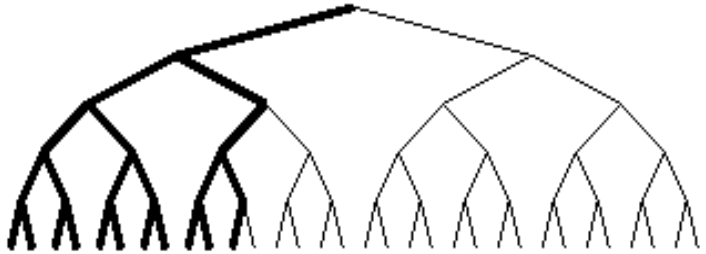
E.g. find variable with maximum coefficient in a symbolic expression:

```
:- lib(linearize).

find_max_weight_variable(ObjectiveExpr, X) :-
        linearize(ObjectiveExpr, [_|Monomials], _),
        sort(1, >, Monomials, [MaxCoeff*X|_Others]).
```

Tree Search
Predefined incomplete strategies (1)

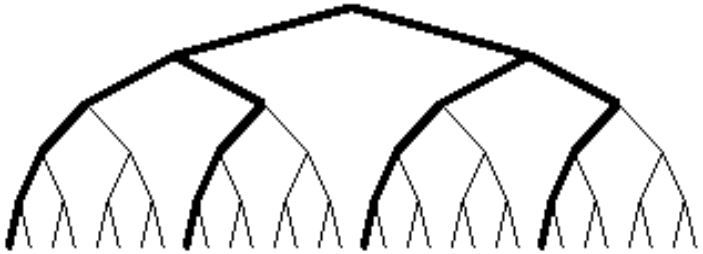
search(List, VarIndex, VarSelect, ValSelect, SearchMethod, Options)

Bounded-backtrack search:



bbs(10)

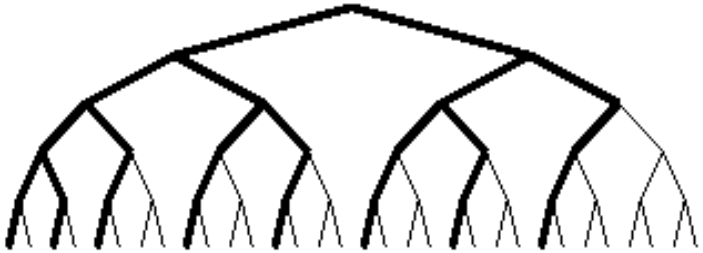
Depth-bounded, then bounded-backtrack search:



dbs(2, bbs(0))

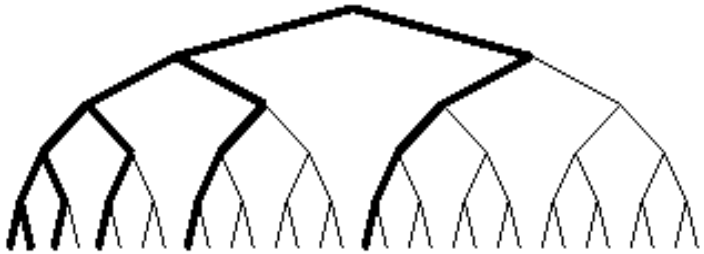
Tree Search
Predefined incomplete strategies (2)

Credit-based search:



credit(16)

Limited Discrepancy Search:



lds(1)

Tree Search
Limited Discrepancy Search

User-defined LDS straightforward to program:

```
lds_labeling(AllVars, MaxDisc) :-
        ( fromto(Vars, Vars, RestVars, []),
          fromto(MaxDisc, Disc, RemDisc, _)
        do
           select_variable(X, Vars, RestVars),
           once select_value(X, Value),
           (
               X = Value, RemDisc = Disc
           ;
               Disc > 0, RemDisc is Disc-1, X #\= Value, indomain(X)
           )
        ).
```

Tree search
Instrumentation, e.g. counting backtracks

```
:- local variable(backtracks), variable(deep_fail).

count_backtracks :-
        setval(deep_fail, false).
count_backtracks :-
        getval(deep_fail, false),
        setval(deep_fail, true),
        incval(backtracks),
        fail.


labeling(AllVars) :-
        ( foreach(Var, AllVars) do
            count_backtracks, % before choice
            indomain(Var)
        ).
```

Properties:

- Shallow backtracking is not counted
- Perfect heuristics leads to backtracks = 0
- Easy to insert in search

Tree search
How to Shave

For finite domains:

```
shave(X) :-
        findall(X, indomain(X), Values),
        X :: Values.
```

E.g. sudoku solvable with ac-alldifferent and shaving – no deep search needed [Simonis].

For continuous variables, we shave off regions from the bounds and iterate until fixpoint:

```
squash(Xs, Precision, LinLog) :-
        …
        ( X $>= Split -> true ; X $=< Split ),
        …
```

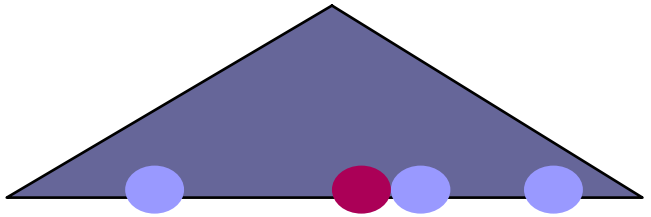
Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do LS
- How to use LP/MIP
- How to do hybrids
- How to visualise

Search Optimization

- Branch-and-bound method
  finding the best of many solutions
  without checking them all

```
:- lib(branch_and_bound).
```



- Strategies:
  Continue – after solution, continue with new bound
  Restart - after solution, restart with new bound
  Dichotomic – search by partitioning the cost space

- Other options:
  Initial cost bounds (if known)
  Minimum improvement (absolute/percentage) between solutions
  Timeouts

Search
Optimization

- Search code for all (or many) solutions can simply be wrapped into the optimisation primitive:

```
setup_constraints(Vars),
bb_min( labeling(Vars), Cost, Options)
```

- The branch-and-bound routine is solver independent
  - Finite and continuous domains
  - LP/MIP
  - Local Search

Tree Search
Optimization with LP solver

```
:- lib(eplex), lib(branch_and_bound).

main :-  ...
         <setup constraints>
         IntVars = <variables that should take integral values>,
         Objective = <objective function>,
         ...
         Objective $= CostVar,
         eplex_solver_setup(min(Objective), CostVar, [], [bounds]),
         ...
         bb_min( mip_search(IntVars), CostVar, _).

mip_search(IntVars) :-
         ...
         eplex_var_get(X, solution, RelaxedSol),
         Split is floor(RelaxedSol),
         ( X $=< Split) ; X $>= Split+1 ),                    % choice
         ...
```

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do LS
- How to use LP/MIP
- How to do hybrids
- How to visualise

Symmetry Breaking

ECLiPSe currently comes with 3 libraries which implement SBDS and SBDD symmetry breaking techniques:

- lib(ic_sbds)
- lib(ic_gap_sbds)
- lib(ic_gap_sbdd)

The last two interface to the GAP system (www.gap-system.org) for the group theoretic operations.

Symmetry Breaking with lib(ic_sbds)

```
queens(Board) :-
    …
    sbds_initialise(Board, 2,
        [r90(Board, N), r180(Board, N), r270(Board, N),
         rx(Board, N), ry(Board, N), rd1(Board, N), rd2(Board, N)],
        #=, []),

    search(Board, 0, input_order, sbds_indomain, sbds, []).
```

```
r90(Matrix, N, [I,J], Value, SymVar, SymValue) :-   % 90 deg rotation
     SymVar is Matrix[J, N + 1 - I],
     SymValue is Value.
…
rd2(Matrix, N, [I,J], Value, SymVar, SymValue) :-   % d2 reflection
     SymVar is Matrix[N + 1 - J, N + 1 - I],
     SymValue is Value.
```

Symmetry Breaking with lib(ic_gap_sbds)

```
queens(Board) :-
    . . .
    sbds_initialise(Board, [rows,cols], values:0..1,
          [symmetry(s_n, [rows], []), symmetry(s_n, [cols], [])],
          []),

    search(Fields, 0, input_order, sbds_indomain, sbds, []).
```

Uses compact symmetry expressions [Harvey et al, SymCon, CP'03]

s_n	**index permutation in 1 dimension**
cycle	**index rotation in 1 dimension**
reverse	**index reversal in 1 dimension**
r_4	**rotation symmetry of the square in 2 dimensions**
d_4	**full symmetry of the square in 2 dimensions**
gap_group(F)	
function(F)	
table(T)	

Overview

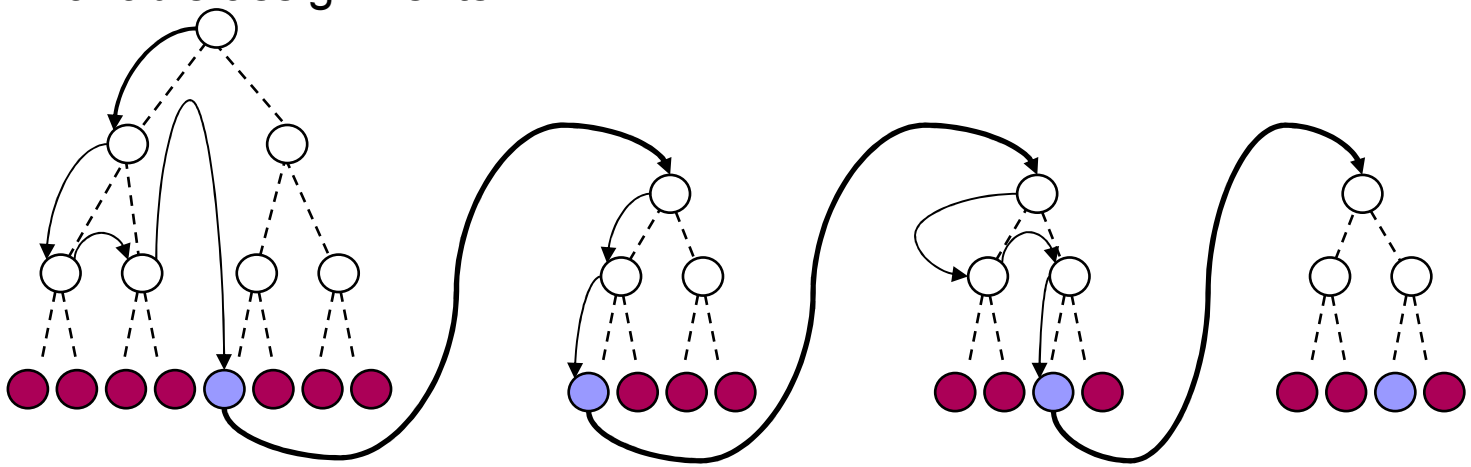
- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do Local Search
- How to use LP/MIP
- How to do hybrids
- How to visualise

Tree Search
With Local Search Flavour

- E.g. Shuffle Search

  tree search within subtrees

  “local moves” between trees, preserving part of the previous solution’s variable assignments



- Pesant & Gendreau, Neighbourhood Models

Tree Search with Local Search Flavour
Restart with seeds, heuristics, limits

Jobshop example (approximation algorithm by Baptiste/LePape/Nuijten)

```
init_memory(Memory),
bb_min((
        ( no_remembered_values(Memory) ->
            once bln_labeling(c, Resources, Tasks),        % find a first solution
            remember_values(Memory, Resources, P)
        ;
            scale_down(P, PLimit, PFactor, Probability),  % with decreasing probability
            member(Heuristic, Heuristics),                % try several heuristics
            repeat(N),                                    % several times
            limit_backtracks(NB),                         % spending limited effort
            install_some_remembered_values(Memory, Resources, Probability),
            bb_min(
                  bln_labeling(Heuristic, Resources, Tasks),
                  EndDate,
                  bb_options{strategy:dichotomic}
            ),
            remember_values(Memory, Resources, Probability)
        )
    ),
    EndDate, Tasks, TasksSol, EndApprox,
    bb_options{strategy:restart}
).
```

Local Search
lib(tentative)

- Tentative values

  ```
  X::1..5, X tent_set 3
  ```

  In addition to other attributes (e.g. domain).
  Tentative value can be changed freely (unlike domain)
  Corresponds to incremental variables in Comet

- Data-driven computation with tentative values
  Suspend until tentative value changes, then execute

- Constraints
  Monitored for degree of violation, assuming tentative values

- Invariants

  ```
  Z tent_is X+Y
  ```

  Update tentative value of Z *whenever* tentative value of X or Y changes (automatic and incremental)

Local Search lib(tentative)

Constraint model [Comet]:

```
:- lib(tentative_constraints).

queens(N, Board) :-
    dim(Board, [N]),                          % make variables
    tent_set_random(Board, 1..N),             % init tentative values

    dim(Pos, [N]),                            % aux arrays of constants
    ( foreacharg(I,Pos), for(I,0,N-1) do true ),
    dim(Neg, [N]),
    ( foreacharg(I,Neg), for(I,0,-N+1,-1) do true ),

    CS :~ alldifferent(Board),                % setup constraints ...
    CS :~ alldifferent(Board, Pos),           % ... in conflict set CS
    CS :~ alldifferent(Board, Neg),

    cs_violations(CS, TotalViolation),        % search part
    steepest(Board, N, TotalViolation),

    tent_fix(Board).                          % instantiate variables
```

Local Search lib(tentative)

Search routine:

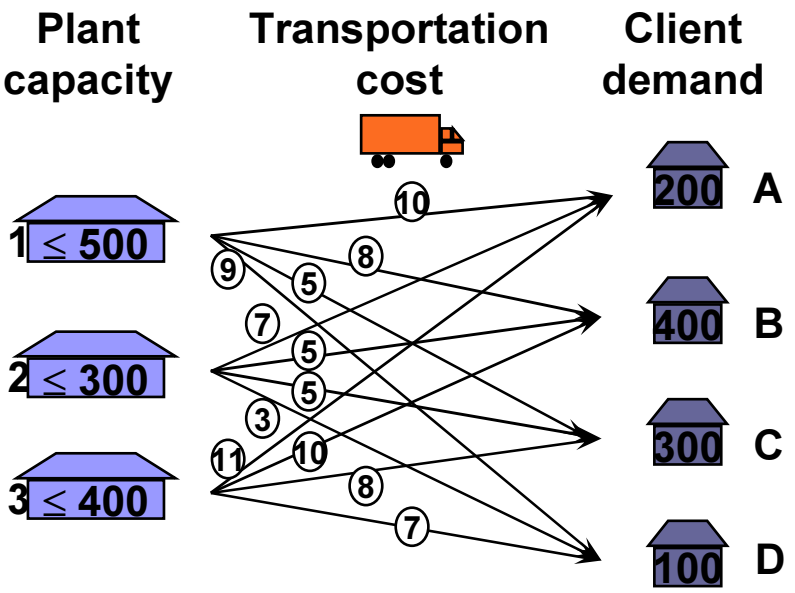
```
steepest(Board, N, Violations) :-
    vs_create(Board, Vars),                    % create variable set
    Violations tent_get V0,                    % initial violations
    SampleSize is fix(sqrt(N)),                % neighbourhood size
    (
       fromto(V0,_V1,V2,0),                    % until no violations left
       param(Vars,N,SampleSize,Violations)
    do
       vs_worst(Vars, X),                      % get a most violated variable
       tent_minimize_random(                   % find a best neighbour
          (                                    % nondeterministic move generator
              random_sample(1..N,SampleSize,I),
              X tent_set I
          ),
          Violations,                          % violation variable
          I                                    % best move-id
       ),
       X tent_set I,                           % do the move
       Violations tent_get V2                  % new violations
    ).
```

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to do Local Search
- How to use LP/MIP
- How to do hybrids
- How to visualise

Mathematical Programming
Solving with Linear Programming

```
:- lib(eplex).

solve(Vars, Cost) :-
    model(Vars, Obj),
    eplex_solver_setup(min(Obj)),
    eplex_solve(Cost).


model(Vars, Obj) :-
    Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
    Vars :: 0..inf,
    A1 + A2 + A3 $= 200,
    B1 + B2 + B3 $= 400,
    C1 + C2 + C3 $= 300,
    D1 + D2 + D3 $= 100,
    A1 + B1 + C1 + D1 $=< 500,
    A2 + B2 + C2 + D2 $=< 300,
    A3 + B3 + C3 + D3 $=< 400,
    Obj =
        10*A1 + 7*A2 + 11*A3 +
         8*B1 + 5*B2 + 10*B3 +
         5*C1 + 5*C2 +  8*C3 +
         9*D1 + 3*D2 +  7*D3.
```



Mathematical Programming
Features of eplex

- Support for: COIN/OSI, CPLEX, Xpress MP
- LP, MIP, QP, MIQP as supported by the solver
- Adding constraints and resolving problem (remove constraints on backtracking)
- Data driven/event triggering of solver
- Encapsulated modifications of problem
- Multiple problem instances
- Options to solver, e.g. changing solving method, presolve, time-outs…
- Support for column generation
- Cut pools

Mathematical Programming Encapsulated modifications: eplex_probe

- Allow problem to be modified temporarily and (re)solved
  - Change/perturb objective
  - Change variable bounds, RHS coefficients,
  - Relax integers constraints….
- Modification is encapsulated into the probe predicate, e.g.
  - What if the transportation cost from plant 1 to warehouse A is increased from 10 to 20?
  - ....

```
eplex_probe([perturb_obj([A1:10])], NewCost)
```

or

```
eplex_probe([min(20*A1+7*A2+11*A3 ...)], NewCost)
```

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to use LP/MIP
- How to do LS
- How to do hybrids
- How to visualise

Solving a Problem with Multiple Solvers

- Real problems comprise different subproblems
  - Different solvers/algorithms suit different subproblems
- Global reasoning can be achieved in different ways
  - Linear solvers reason globally on linear constraints
  - Domain solvers support application-specific global constraints
- Solvers complement each other
  - Optimisation versus feasibility
  - New and adapted forms of cooperation
    - (e.g. Linear relaxation as a heuristic)

Co-operating Solvers in ECLiPSe

- Modelling of different (sub)problems for different solvers can use the same logical variables:

```
ic: (X #>= 3), eplex: (X + Y $=< 3.0)
```

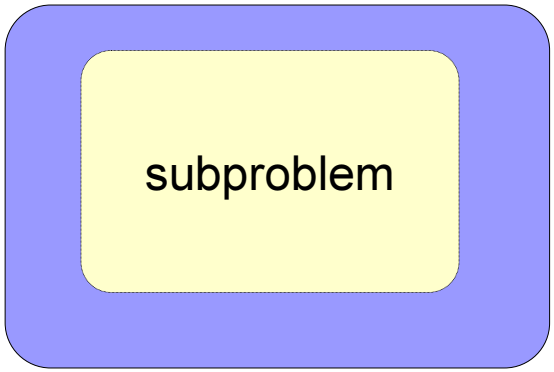
- Common Solver Interface – same syntax for constraints in different solvers

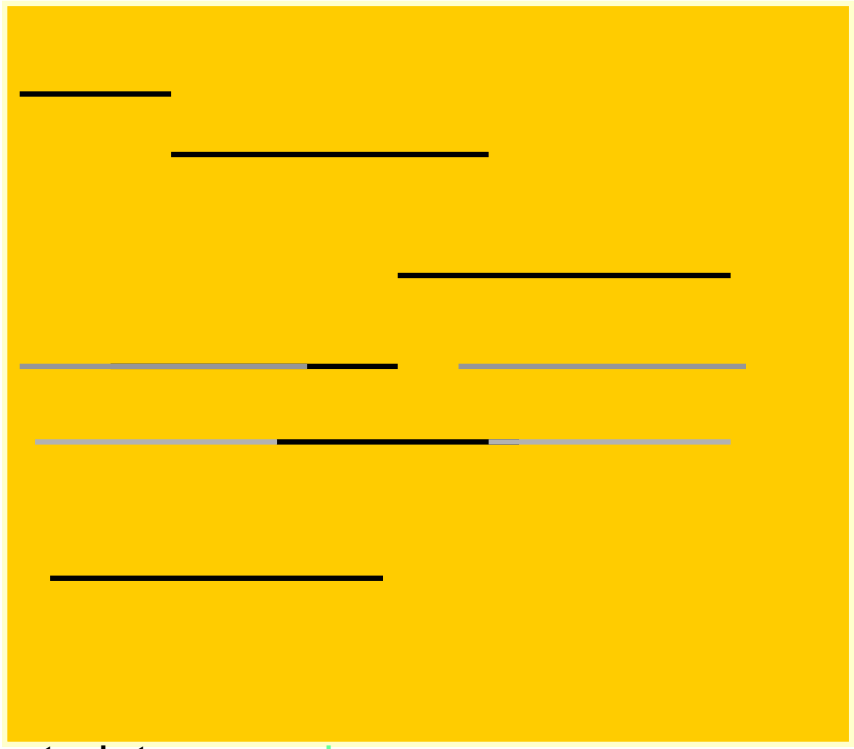
```
ic: (X $=< 3), eplex: (Y $=< 3)
[ic, eplex]: (X + Y $=:= Z + 3)
Solver: (X+Y $>= 3)
```

Example: Bridge Scheduling Problem

- Linear precedence + distance constraints (min/max time between tasks)
- Non-linear resource usage constraints: nonoverlap of tasks using the same resource
- Find optimal solution (minimise)
- Hybrid solution for problem using probe backtrack search – note real problems that benefits from hybrid will be more complicated.

Probe Backtrack Search





Conflicting constraints for main problem (monitored)

add constraint to subproblem to resolve one conflict (remove overlap)

resolve subproblem with added constraints

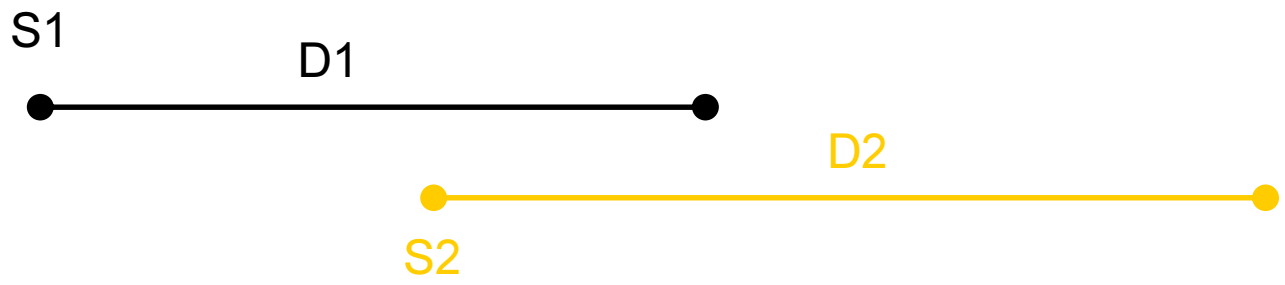
More than one way to resolve constraint ->search

Probe Backtrack

- Use a “probe” to solve a subproblem of the full problem.
  - Subproblem can be a problem class suitable for specialised solver
- Constraints not sent to subproblem are monitored for violation
- Solution values from probe used as tentative solution values for main problem
- If no monitored constraints are violated, we have a feasible solution to main problem, otherwise:
- Repair solution by selecting a violated constraint, and add constraints to subproblem to remove the violation, solve subproblem again.
  - added constraints are specialised version of the violated constraint, suitable for subproblem.
  - There may be > 1 way to remove violation – choice
- Use branch and bound search to obtain optimal solution.

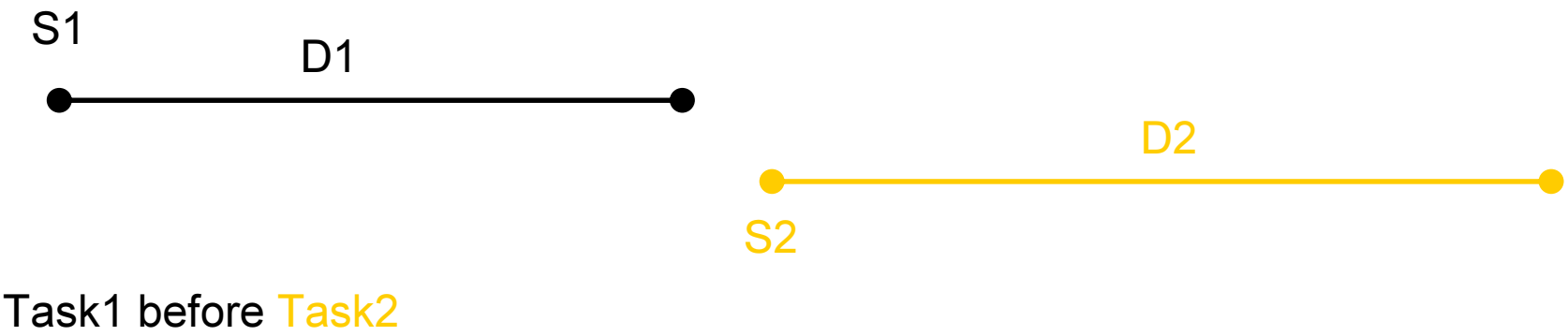
Bridge Problem: Probe Backtrack

- Probe with eplex: MP solver for linear problem
- Subproblem is problem without the non-linear nonoverlap constraint
- Repair a violated constraint by pushing apart overlapping task pairs that use the same resource:



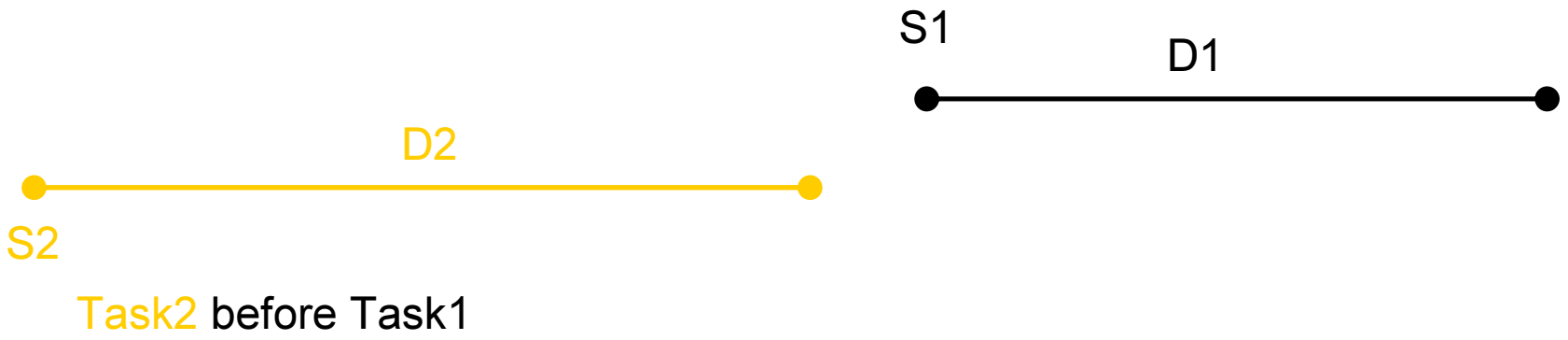
Bridge Problem: Probe Backtrack

- Probe with eplex: MP solver for linear problem
- Subproblem is problem without the non-linear nonoverlap constraint
- Repair a violated constraint by pushing apart overlapping task pairs that use the same resource:

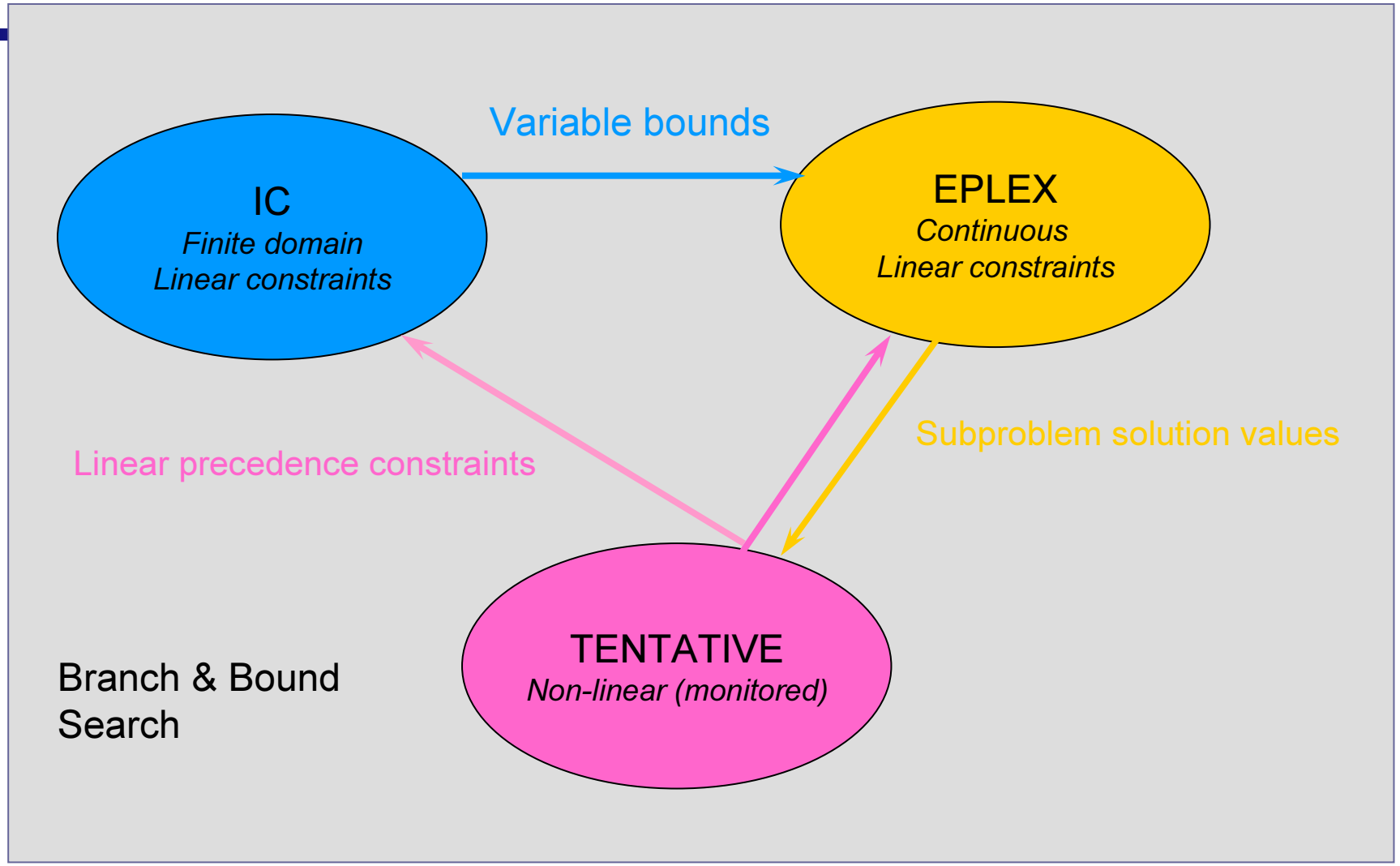


Bridge Problem: Probe Backtrack

- Probe with eplex: MP solver for linear problem
- Subproblem is problem without the non-linear nonoverlap constraint
- Repair a violated constraint by pushing apart overlapping task pairs that use the same resource:



Information transfer



Bridge Probe Backtrack Example

```
:- lib(tentative), lib(eplex),lib(ic),lib(branch_and_bound).
:- local struct(task(start,duration,need,use)).
go(End_date) :-
        Tasks = [PA,….],
        PA = task{duration : 0, need : []},
        A1 = task{duration : 4, need : [PA], use : excavator},
    ….
        end_to_end_max(S6, B6, 4),
    ….
        end_to_start_max(A6, S6, 3),
    ….
        start_to_start_min(UE, Start_of_F, 6),
        end_to_start_min(End_of_M, UA, -2),
        tasks_starts(Tasks,Starts),
        ic:integers(Starts),
        (foreach(task{start:Si,duration:_Di,need:NeededTasks}, Tasks) do
            [ic,eplex]:(Si $>= 0),
        ),
         (foreach(task{start:Sj,duration:Dj},NeededTasks), param(Si) do
                [ic,eplex]:(Si $>= Sj+Dj)                    )
        ),
        tent_set_all(Starts, 0),
        disjunct_setup(Tasks, CS),
        eplex_solver_setup(min(End_date), End_date, [sync_bounds(yes)/].
                new_constraint, deviating_bounds, post( post_eplex_solve )]),
        minimize((
                repair_label(CS),
                tent_fix(Starts) ), End_date).
post_eplex_solve :-
      eplex_get(vars, Vs),
      eplex_get(typed_solution, Vals),
      ( foreacharg(V,Vs),  foreacharg(Val0, Vals) do
        Val is fix(round(Val0)), V tent_set Val    ).

start_to_start_min(task{start:S1}, task{start:S2}, Min) :-
        [ic,eplex]:(S1+Min $=< S2).
end_to_end_max(task{start:S1,duration:D1}, task{start:S2,duration:D2}, Max) :-
        [ic,eplex]:(S1+D1+Max $>=S2+D2).
end_to_start_min(task{start:S1,duration:D1}, task{start:S2}, Min) :-
        [ic,eplex]:(S1+D1+Min $=< S2.
end_to_start_max(task{start:S1,duration:D1}, task{start:S2}, Max) :-
        [ic,eplex]:(S1+D1+Max $>= S2.
```

```
disjunct_setup(Tasks, CS) :-
        cs_create(CS, []),
        ( fromto(Tasks, [Task0|Tasks0], Tasks0, []), param(CS) do
            ( foreach(Task1,Tasks0), param(Task0,CS) do
                Task0 = task{start:S0, duration:D0, use:R0},
                Task1 = task{start:S1, duration:D1, use:R1},
                 ( R0 == R1 ->
                        CS :~ nonoverlap(S0,D0,S1,D1) alias tasks(Task0, Task1)

                 ;
                        true
                 ) ) ).
```

```
nonoverlap(Si,Di,Sj,_Dj) :-
        [ic,eplex]: (Sj $>= Si+Di).
nonoverlap(Si,_Di,Sj,Dj) :
-       [ic,eplex]: (Si $>= Sj+Dj).
repair_label(CS) :-
        ( cs_all_worst(CS, [tasks(Task0, Task1)|_]) ->
            Task0 = task{start:S0, duration:D0},
            Task1 = task{start:S1, duration:D1},
            nonoverlap(S0,D0,S1,D1),
            repair_label(CS)
        ;
            true % no overlaps
        ).
```

Bridge Example code

- Use ic and eplex, tentative and branch_and_bound libraries
- Use task structure to represent tasks:

  :- local struct(task(start,duration,need,use)).
- Constraint set up, e.g. :

```
start_to_start_min(task{start:S1}, task{start:S2} , Min) :-
        [ic,eplex]:(S1+Min $=< S2).
```

Setup of nonoverlap conflict monitoring

Handle to a conflict set

```
cs_create(CS, []),
(  fromto(Tasks, [Task0|Tasks0], Tasks0, []), param(CS)
        ( foreach(Task1,Tasks0),  param(Task0,CS) do
            Task0 = task{start:S0, duration:D0, use:R0},
            Task1 = task{start:S1, duration:D1, use:R1},
            ( R0 == R1 ->
                    % tasks uses same resource – monitor for overlap
                    CS :~ nonoverlap(S0,D0,S1,D1) alias tasks(Task0,Task1)
            ;
                    true
            )
        )
)
```

alias defines how monitored constraint is returned

Add constraint to conflict set: monitored version called to check for violation

Setup of eplex probe

```
……..
eplex_solver_setup(min(Endtask), Endtask,
        [  sync_bounds(yes) ],
        [  new_constraint,
           deviating_bounds,
           post( post_eplex_so
        ]
 ),
 …..

post_eplex_solve :-
    eplex_get(vars, Vs),
    eplex_get(typed_solution, Vals),
    (  foreacharg(V,Vs),  foreacharg(Val0,
         Val is fix(round(Val0)),
         V tent_set Val
     ).
```

Update bounds before solve

Trigger solver when new constraints are

…and when bounds change to exclude existing eplex solution

Update tentative value with the eplex solution values (as integer values). Violation of monitored constraints is checked for

Repair of violated constraints

Use branch and bound search to find optimal solution

```
:- lib(bra

    …
    minimize(
      (     repair_label(CS),
            tent_fix(Starts)
      ), Cost)

repair_label(CS) :-
    ( cs_all_worst(CS, [tasks(Task0, Task1)|_
            Task0 = task{start:S0, duration:D0
            Task1 = task{start:S1, duration:
            nonoverlap(S0,D0,S1,D1),
            repair_label(CS)
    ;
            true % no violated nonoverlap
    ).
```

Repair the violated constraint in Instantiated to feasible solution

Impose nonoverlap constraint to remove conflict

Disjunctive nonoverlap constraint

```
nonoverlap(Si,Di,Sj,_Dj) :-
        [ic,eplex]:Sj #>= Si+Di.
nonoverlap(Si,_Di,Sj,Dj) :-
        [ic,eplex]:Si #>= Sj+Dj.
```

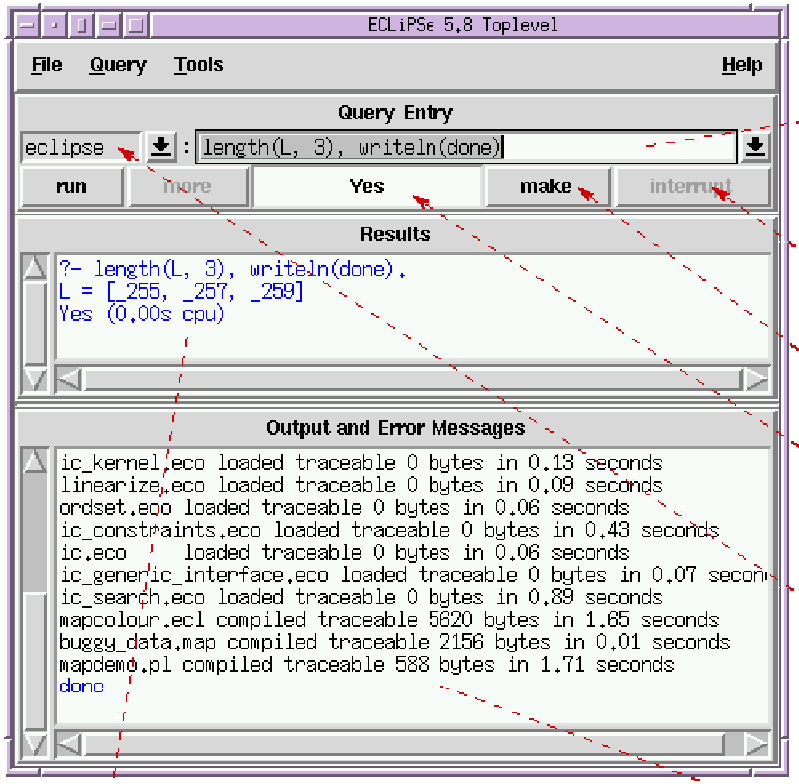
Added new ordering constraint between the two tasks. Create a choice-point in the search for the two orderings

Post ic constraint first to allow failure before solving problem with eplex

Overview

- How to model
- How to use solvers
- How to prototype constraints
- How to do tree search
- How to do optimization
- How to break symmetries
- How to use LP/MIP
- How to do LS
- How to do hybrids
- How to visualise

Tools
TkEclipse



Query entry window

Type in query here
History mechanism:
1. up/down arrow keys
2. press arrow box for history list
3. right–click for history list (with duplicates)

Interrupt button

Press to interrupt program execution
(Disabled if no program is running)

Make button

Press to recompile changed programs

Query status window

Displays status of last query

Current module

Shows current module for query entry
Change module by pressing down arrow box and select from list (new module must be created from 'New module' option of File menu

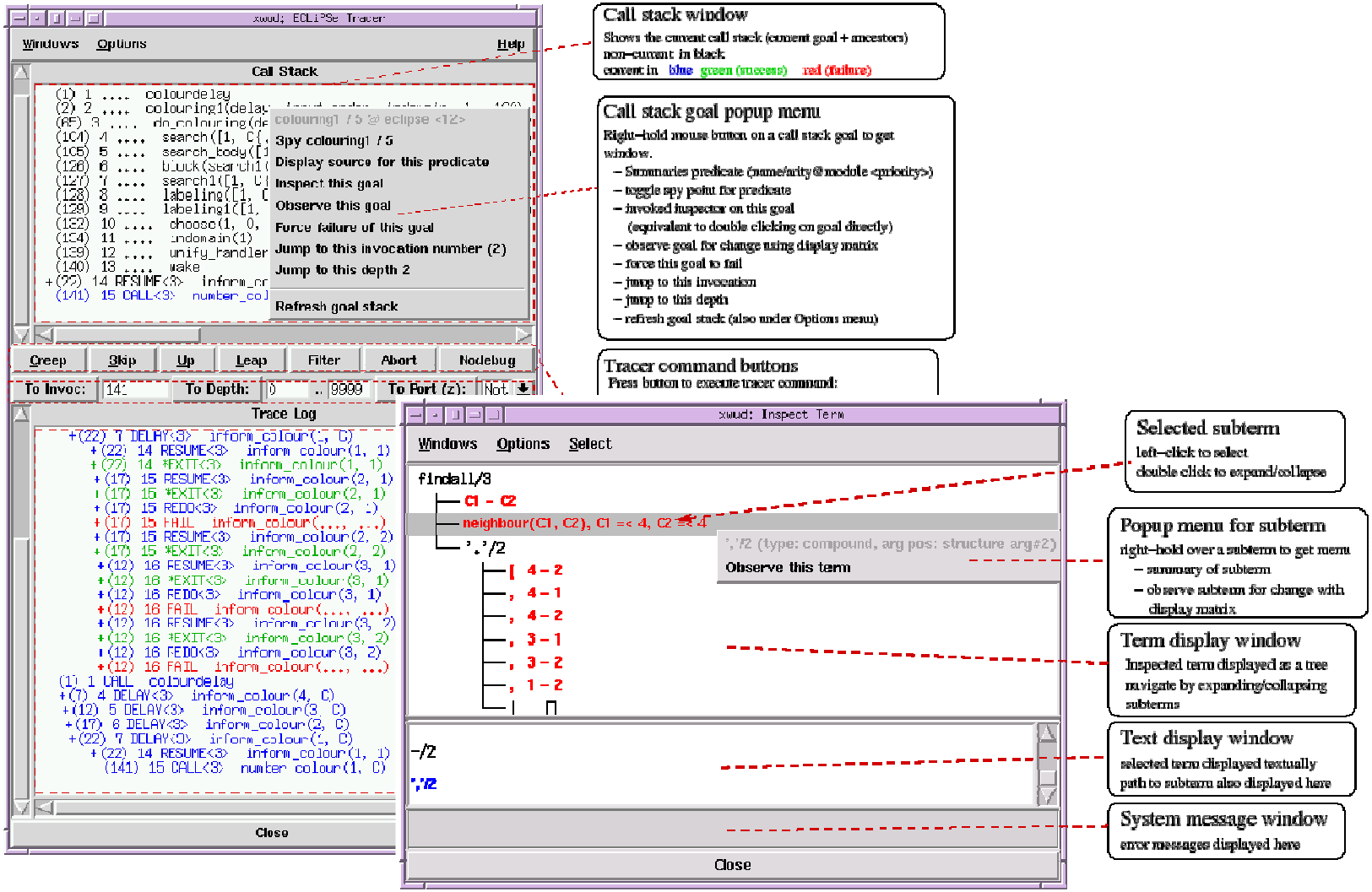
Output window

Output from program appears here
– most current output in blue
– old output in black
– error output in red
– warning output in orange

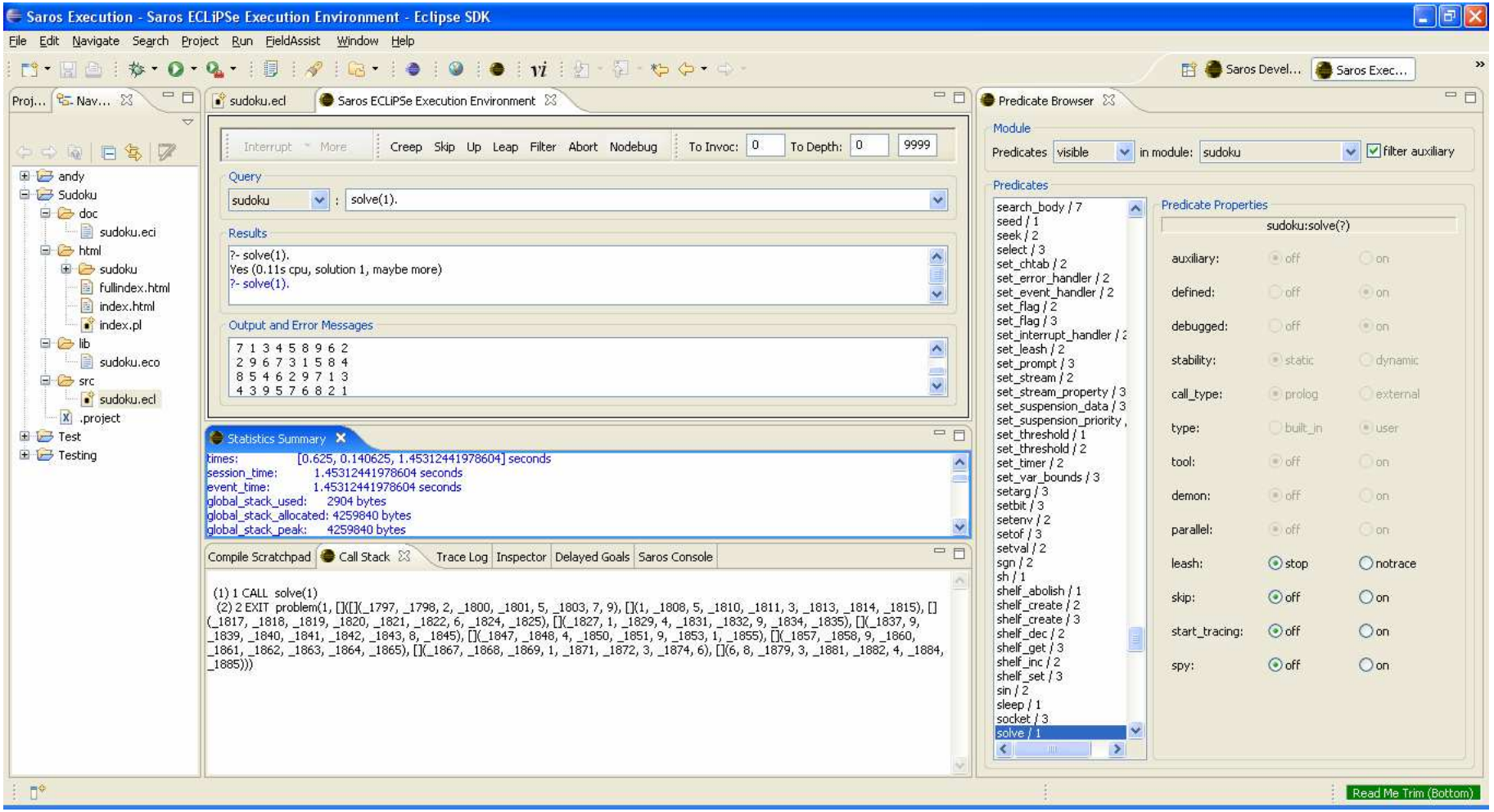
Results window

Query, bindings to query, execution status of query appears here
– most recent query in blue
– older queries in black

Tools
Tracer and Data Inspector



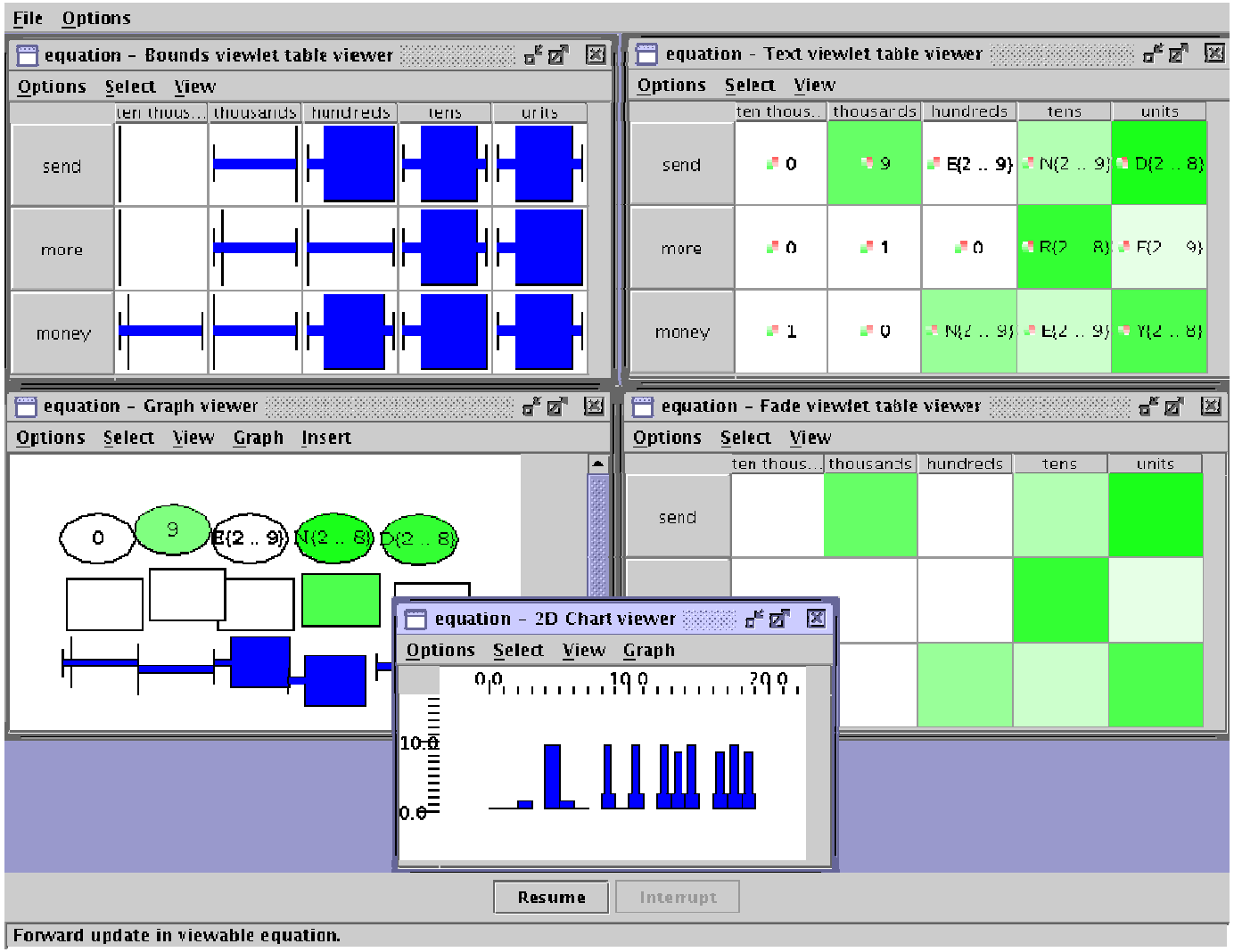
Tools
Saros – ECLiPSe/eclipse integration



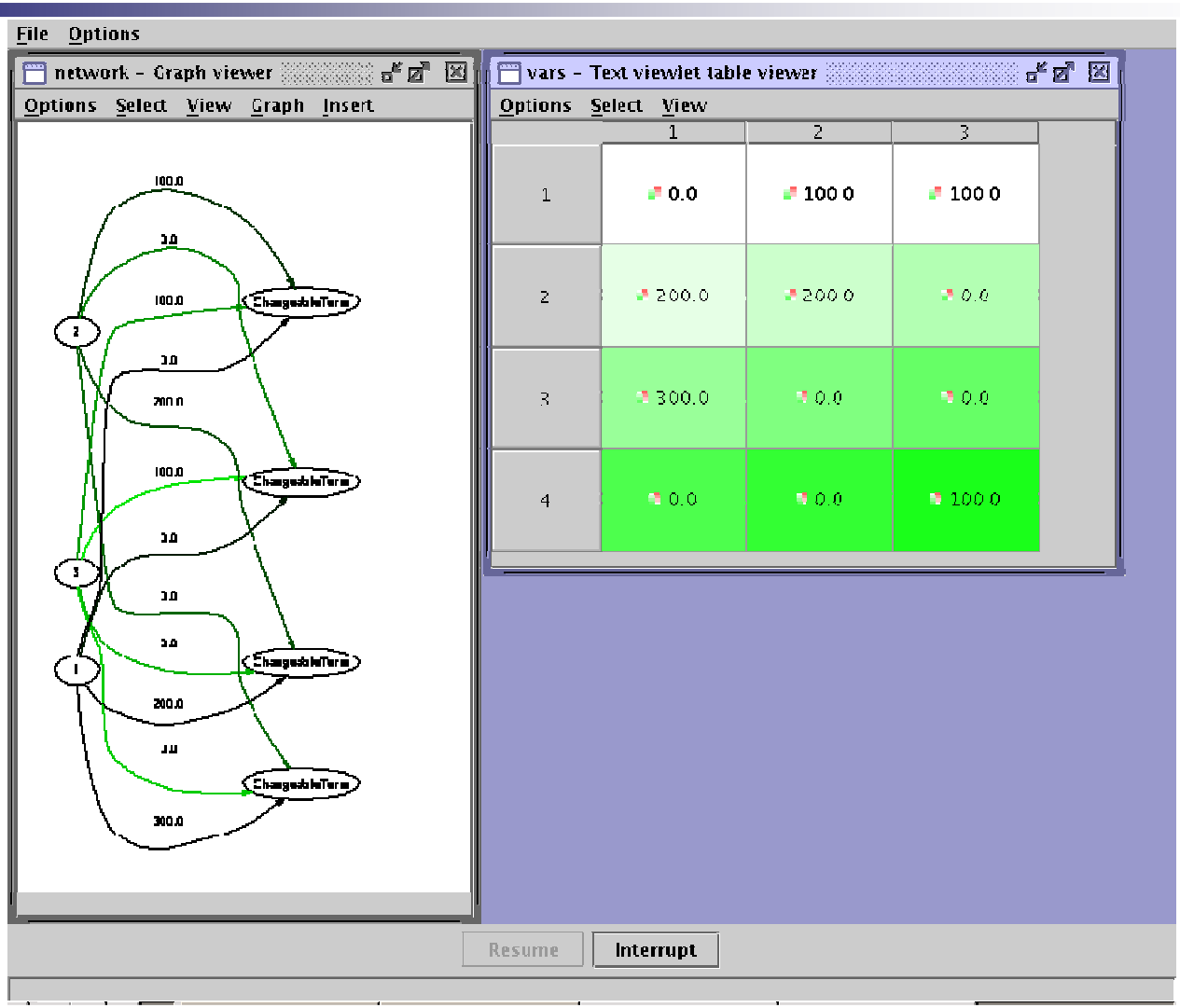
Tools
Visualisation

```
:-lib(ic).
:-lib(viewable).

sendmore1(Digits) :-
  Digits = [S,E,N,D,M,O,R,Y],
  Digits :: [0..9],
  viewable_create(equation, Digits),
  Carries = [C1,C2,C3,C4],
  Carries :: [0..1],
  alldifferent(Digits),
  S #\= 0,
  M #\= 0,
  C1          #= M,
  C2 + S + M #= O + 10*C1,
  C3 + E + O #= N + 10*C2,
  C4 + N + R #= E + 10*C3,
       D + E #= Y + 10*C4,
  lab(Carries),
  lab(Digits).
```

Tools
Visualisation (arrays)



Tools
Visualisation (graphs)



Ongoing work

- Evolve strengths
  - High level scripting and solver cooperation
  - More interfaces to good solvers (Gecode, MiniSat, …)
- System Engineering
  - New compiler
  - Runtime improvements
- Development Environment
  - Saros (ECLiPSe/eclipse)

Resources

- Books
  - Constraint Logic Programming using ECLiPSe
    - Krzysztof Apt & Mark Wallace, Cambridge University Press, 2006.
  - Programming with Constraints: an Introduction
    - Kim Mariott & Peter Stuckey, MIT Press, 1998.

- ECLiPSe is open-source (MPL)
  - Main web site www.eclipse-clp.org
  - Tutorial, papers, manuals, mailing lists
  - Sources at www.sourceforge.net/eclipse-clp